A Comparative Study of Moded and Modeless Text Editing By Experienced Editor Users

Merle F. Poller and Susan K. Garter
Bell Laboratories
Piscataway, New Jersey

As interactive text editing systems become more and more pervasive on the job, at school, and in the home, the necessity for good human engineering in the design of such interactive systems becomes increasingly apparent. Cognitive psychology, ergonomics, and practical experience have already provided us with many guidelines for developing good human-computer interfaces; however, the rapid and sophisticated advances that have occurred in editor software and hardware have limited the collection of human performance data on editor use. Such data will help us to expand existing guidelines and apply them to their fullest.

In the area of text editing, one question for which human performance data will be useful is the question of whether moded or modeless editing is preferable. Modes refer to the editor's interpretation of the user's input; there is a mode change each time the interpretation of the same input changes. An editor is modeless if its interpretation of the same user input is always the same.

The issue we are concerned with in this paper is not that of modes in general, but rather the more specific question of how editors should handle text insertions. In this context, moded editing means that the editor user must enter a special command before text is inserted and another special command to end the text insertion and return to the command mode. Ordinary printing characters typed while in an insertion mode are entered as text. The same characters entered while the editor is in the command mode are treated as commands. Modeless editing is different; ordinary characters are entered directly as text. There are no special commands required to enter or stop entering text.

In modeless editing, commands must be entered with special keys. Sometimes the editor is part of a system that has a terminal with many special function keys and perhaps a mouse to move the cursor and perform other pointing and selecting functions [Tesl81]. In other cases, the editor may be a general purpose one designed to work on a standard 24 line by 80 character CRT. In the latter cases, all commands must be, or at least begin with, control or escape characters.

Some potential advantages of modeless editors are fairly obvious. First, naive computer users can use a modeless editor the way they use a typewriter; they start with a blank page (screen), begin typing, and the text appears. Second, modeless editor users learn and use fewer commands, since no commands are needed to enter and stop entering text. Third, the absence of insertion modes in modeless editors should preclude "moded errors," which occur when people mistakenly enter text while in a command mode or commands while in an insertion mode [Good81, Newm79]. Other potential advantages may be less obvious. Modeless editing may be faster than moded editing since the presence of modes may cause increases in reaction times [Thim82]. Finally, modeless editing may be more flexible since it doesn't restrict the functions people can use [Meyr82]. For example, people who use modeless editing have told us that it allows them to compose more freely at the terminal.

But there are tradeoffs. To get these modeless benefits, we give up the capability of using ordinary keys to issue commands. What effect does this loss have? For modeless editors on standard CRTs, all commands must at least begin with control or escape characters. In contrast, moded editors on the same terminals can use all the ordinary alphanumeric characters in addition to the characters available to the modeless editors. This greater number of available characters means that moded command languages can choose more mnemonic names, can have more single character names, and can provide easy-to-type unshifted characters for common, frequently used commands.

Clearly, the tradeoffs between moded and modeless editing depend on both the users and the terminal type. The research we are reporting had several goals, one of which was to explore moded and modeless editing by experienced, non-clerical editor users working on standard terminals. Ideally, we would have liked to study this question with specially designed editors that differed only in whether they were moded and with subjects assigned at random to these editors; however, we found that if we wanted to look at experienced editor users, we had to look at pre-existing groups and editors. Luckily, we had available two screen editors for UNIX* operating systems, in daily use by technical staff, and felt we could control many extraneous variables such as system differences and some crucial editor differences.

* UNIX is a trademark of Bell Laboratories.

These two editors were *vi* [Joy81], written at the University of California at Berkeley, and *emacs*, written for a UNIX operating system by W. Montgomery. This *emacs* was patterned after the *emacs* [Stal79] written for the PDP-10 systems at the Massachusetts Institute of Technology. Although it differs from the MIT version, especially in its macro language, its default command interface is similar.

Both *emacs* and *vi* are powerful full-screen editors that process input by character rather than by line. Both use short (one- and two-character) command names. They differ in features that we did not look at or use because they can be used equally well with moded and with modeless editors. These differences included features such as extensibility, multiple windows, options, and programming aids. Of interest to us was a major difference between the two editors that affects even the simplest of editing jobs: namely that *vi* is moded, requiring a command such as "a" to enter the insertion mode and the command <escape> to return to the command mode, while *emacs* is modeless, with ordinary characters entered as text, and commands entered with control and escape characters.

METHODS AND PROCEDURES

Subjects

We began by searching the UNIX operating system accounting records at Bell Laboratories in Piscataway for people who used the commands *vi* and *emacs*. This method of identifying subjects allowed us to limit our choice to the population that was really using these editors in their work. Next, using telephone interviews, we selected people who also met our other requirements, which were that (1) they held a technical rather than clerical position, (2) they had at least five months of experience with their editor, (3) they were familiar with and used one of the Hewlett-Packard terminal keyboards, and (4) they used their editor to both enter and edit their own English language documents.

We collected data from eight *vi* and ten *emacs* users. We eliminated the data from two of the *emacs* users: one was not familiar with the terminal; the other was given incorrect procedures in the performance part of the study.

Results of a brief questionnaire and a short typing test, both administered at the start of the experimental session, showed that the two groups were similar in job experience, but the *emacs* users were slower typists. All but one participant, a *vi* user, wrote programs with their editors, and most used other editors as well.

Hardware and Software

The two editors were on the same Western Electric computer that was running under UNIX operating system release 4.2. Since editor performance depends on system use, we checked the number of system users and the time to compile a standard C program several times during each session, finding that mean system use was not different for the two groups.

We modified the two editors slightly to record all characters entered by the subjects and the times of entry. A separate check comparing the software measured times with hardware measured times showed that 99% of the recorded times were within one second of the real time.

Our subjects worked on either an HP2621 or HP2645 terminal, whichever they were familiar with. All special function keys and the break key were covered with masking tape, and the subjects were asked not to use them. The subjects began with a practice task that allowed them to become familiar with the set-up and to ask questions about the editors, terminals, and tasks.

Performance Tasks

We decided to look at both fixed editing and free composing since the effects of modeless and moded editing might depend on the type of task being performed. We were given a head start on the fixed editing tasks by the works of Roberts [Robe79] and Roberts and Moran [Robe81], which provide a set of procedures for benchmarking editors. Part of the benchmarking consists of a set of 50 editing subtasks embedded in four documents: a short inter-office memo, two two-page reports, and one chapter from a philosophy book. These 50 subtasks sample the "core" editing tasks that Roberts identified in her doctoral thesis. These subtasks included adding, changing, and deleting text, and also moving and copying text. We used 31 of the 50 subtasks in the same four documents, eliminating 19 subtasks because of time constraints. We added a fifth document, a Bell Laboratories magazine article, with editing subtasks of short inserts, deletes, and changes applied to the beginning, middle, and ends of words, sentences, and lines. This new task also had several global activities.

The free composition task, patterned after tasks used by Allen [Alle81], asked the subjects to write a book or movie review. The instructions encouraged the subjects to edit their compositions.

After completing the performance tasks, subjects filled in a questionnaire that asked them to rate the difficulty of some of the subtasks and to rate their editor along several dimensions.

RESULTS AND DISCUSSION

Overall Editing Performance

There are two major criteria of overall editing performance: how good the final product is, and how long it takes to get that final product.

To identify errors in the final files, we used **diff**, a UNIX operating system command, to print out all differences between correct final files and the subjects' final files. We counted all differences as errors, except for differences resulting from different line breaks, extra lines between paragraphs, and different numbers of blanks. The error count was the count of subtasks that had at least one error. We transformed these error counts by taking the square root of the sum of the error count and 0.5. This transformation, when applied to scores such as number of errors, makes the error variances more equal [Wine71]. To see if the number of errors was related to typing skills, we correlated the transformed error scores with the square root of the typing speeds within each group. The low correlation coefficients obtained (+.14 for the *emacs* group and +.03 for the *vi* group) suggest that differences in errors did not result from differences in typing skills. We then performed an analysis of variance on the transformed error scores with editor as a grouping factor and task as a repeated measure. This analysis showed the *vi* group left fewer errors in the final file than did the *emacs* group ($F(1,14) = 7.18$, $p = .02$); the *vi* group left errors in 5% of the subtasks while the *emacs* group left errors in 11% of the subtasks.

A breakdown of the errors showed that some were simple uncorrected typographical and copying errors (8 for *emacs* and 11 for *vi*), while others were subtasks that were not done (6 for *emacs* and 4 for *vi*). Some of the remaining errors were related to text moves and may result from the way people tended to move text in the two editors rather than from anything related to moded and modeless editing. The remaining errors (33 for *emacs* and 8 for *vi*) were a mixture, with the highest number occurring in subtasks that required a change rather than just an insertion or a deletion.

Why did the editor groups differ in number of errors? We can think of several plausible hypotheses for future research. The first is that the simpler nature of the *vi* command names allowed people to concentrate more on their product and less on their editing. Another possibility is that having modes actually helps users by forcing them to stop and exit before they can continue. Perhaps this enforced pause causes them to check their work more carefully. A third possibility, which would explain the change errors, is that

vi's change command requires that the users specify what will be deleted before they start inserting. *emacs* users have to do an insertion and a separate deletion. They tend to do the insertion first, and may forget to do the deletion to complete the change.

Our other major editing criterion is speed. We did an analysis of variance on the log of the task times. Log transformations, when applied to time data, usually reduce the skewness of the data and make the error variances more equal [Wine71]. The analysis showed a tendency ($p = .06$) for the *emacs* group to take longer than the *vi* group. However, the *emacs* group's typing rate was lower than that of the *vi* group. The keystroke model [Card80] suggests that, all other things being equal, editing time will depend on typing rate. We did not apply Card's model for correction because it is unclear on how to handle repeated characters. In our study, the *emacs* users repeated many characters, especially while moving the cursor. An analysis of covariance, with typing as the covariate, suggested that any time differences between the two groups might result from typing differences. However, this analysis is open to question since we really don't know if editing speed and typing speed are correlated. In our study, they were for the *emacs* group, but not for the *vi* group.

Another way of evaluating editors is to ask users about them. Our subjects rated the ease of performing eight of the editing subtasks on a scale from 1 to 7. Some subtasks were judged harder than others, but there were no significant differences between the editors. In general, the subtasks were judged easy to do.

Perhaps users perceive other differences in their editors. We asked them to rate their editors on the dimensions identified by Dzida, Herda, and Itzfeldt [Dzid78]. The dimensions were ease of learning, editor self-descriptiveness, amount of user control the editor allowed, degree of correspondence between editor behavior and user expectations, degree of extensibility, and tolerance for erroneous user input. Respondents also rated the ease of remembering and the consistency of command names. Each dimension was defined for the subjects by a list of each of the qualities that had had a high rating on that dimension. For example, the rating for degree of extensibility read:

> An editor that is highly extensible:
>
> - Allows the user to expand the command language.
> - Provides system messages at differing levels of detail, dependent on user preference.
> - Allows users to define sets of system functions.
> - Provides shortcuts for expert users.
> - Permits clustering of commands with a new name.

Separate analyses of variance were performed for each question. Two dimensions showed statistically significant differences between groups: the *emacs* group rated their editor more extensible ($F(1,14) = 9.00$, $p = .01$) and more tolerant of erroneous input ($F(1,14) = 5.33$, $p = .04$). Another difference approached statistical significance: the *emacs* group tended to rate their editor as more self-descriptive ($F(1,14) = 3.67$, $p = .08$).

Explanations of two of these differences seem straight-forward. *emacs* is designed to be extensible, and it has more self-documentation and on-line help facilities than does *vi*.

The explanation of *emacs* being more tolerant is less clear, but may result from the first tolerance quality defined, which was that a tolerant editor allows for only partial retyping of erroneously entered commands. The *emacs* incremental search command allows users to delete characters from the search string if no pattern match was found. *vi* does not provide this; if a string is not matched in *vi*, the user must retype the entire string to do the search.

The differences we found may result from editor differences or from group differences. This latter possibility, which exists for any study that uses pre-existing rather than experimentally formed groups, always limits the conclusions that can be drawn from that study. Nevertheless, such studies are often done when experimentally formed groups are not practical; in our case, it was impossible to assign editor-naive people to groups, provide them with equivalent training, and expect them to use the editors equally often for at least five months. However, our study did control many variables that may confound editor studies. The system and speed were the same for the two editors, the editors displayed the same number of lines of text, the subjects used their editors for similar tasks on the job and were familiar with the terminal keyboards they used in the study, and differences in typing speeds were controlled statistically.

Moded Errors

People who like modeless editing point out that modeless editing avoids the errors that occur when people forget what mode they are in. How much of a problem are moded errors for experienced editor users?

To identify moded errors, we wrote two programs, one for *vi* and one for *emacs*, that printed out sequences of command characters and sequences of text characters. We then read the printouts, paying particular attention to the beginnings and endings of these sequences. We defined a moded error as either of the following events:

- Characters intended as commands were entered as text.
- Characters intended as text were treated as commands.

Table 1 reports the total moded error counts by task type. For *vi*, the table also reports the number of times an inserting text mode was entered.

We found that people using the moded editor *vi* did make moded errors. However, we also found that subjects using the modeless editor *emacs* made responses that fit the definition of the first type of moded error. In these cases, the *emacs* subjects clearly intended commands (such as <escape>b or <control>b), but instead entered the corresponding letter (such as b) into the text.

The results in Table 1 suggest that *vi* users may make more moded errors while composing than while doing the fixed editing tasks. One explanation of this difference is that moded errors may become more frequent as the size of the insertion increases [Snee78]. This would explain the higher frequency of moded errors during composition since the insertions during composition were longer than during fixed editing. The *emacs* subjects made few moded errors while composing, probably because most *emacs* errors occurred during search commands, which the *emacs* subjects rarely used while composing.

Table 1. Moded Errors

Editor	Total Moded Errors	Moded Errors by Task Type		
		Copy Text	Fixed Editing	Free Composition
emacs	31	0	29	2
vi*	16 (551)	2 (8)	4(305)	10 (238)

*Note: The numbers in parentheses give a count of how often an inserting text mode was entered in *vi*.

All these errors were rapidly identified and corrected by the subjects. Both *emacs* and *vi* subjects used a delete character command to eliminate unwanted text. For *vi*, when text was typed in the command mode, the text, which was just a few characters, was retyped. If the buffer had been changed, the subjects used *vi*'s undo command to correct it.

Clearly moded errors were infrequent and did not create any significant problems for any of the participants in any of the tasks. This was true even though the *vi* editor did not provide feedback to the user about which mode the user was in.

Why then are moded errors usually regarded as a problem? One possibility is that they create problems for people while they are learning to use the editor. We hope to explore this in our future research.

A second possibility is that moded errors are easier to avoid and to correct in full-screen, character-oriented editors than they are in line-oriented editors in which editing changes are made only after a carriage return is entered. We have some data from users of *ed*, a UNIX operating system line-oriented editor, that agrees with this. If true, people who have learned a moded line-oriented editor may be erroneously generalizing the problems they have had in the past on line-oriented editors to full-screen, real time editors.

Composing With Modeless and Moded Editors

We evaluated the compositions written by the subjects with style and spell, two UNIX operating system commands. The style program provides many measures of writing style that are related to the quality of writing [Cher82, Macd82].

The writing variables we looked at were readability grade level (as measured by the Kincaid formula), number of sentences, number of words, sentence length, word length, percentage of simple sentences, percentage of passive voice sentences, percentage of sentences that began with the subject of the sentence, number of long sentences, number of spelling and typographical errors, and, finally, the total time spent at the terminal. Separate analyses of variance were performed on each of these variables. The *vi* group showed a tendency to write a greater percentage of simple sentences than the *emacs* group. ($F(1,14) = 4.40$, $p = .055$). The other variables were similar for the two groups.

emacs users we interviewed before we began this study told us that they felt they could move about and edit more freely as they wrote. Several of these people had used *vi* or other moded editors for a period of time because they did not have a modeless editor available on their system. They reported feeling constricted by having to exit the inserting text modes to do any serious editing. On the other hand, *vi* users we interviewed were either unaware of this problem or, at worst, regarded it as a minor inconvenience.

We wanted to see if editing with *emacs* while composing did indeed follow a different pattern than editing with *vi*. To do this, we used the programs previously developed for identifying moded errors to identify edits that involved inserting and changing text. This eliminated backspacing and line kills to correct typing and other easily caught mistakes. Next, we read the data files to locate the point at which the first draft was completed and counted the number of edits that occurred before the end of the first draft. Figure 1 shows a plot of the percentage of edits done during the writing of the first draft as a function of total edits.

First, note that although our subjects varied widely in how much editing they did, the total amount of editing done was not related to the editor used (Mann-Whitney, $p = .44$, one-tailed). However, our concern really was with how much editing occurred during the writing of the first draft. Here again, the subjects varied, with some doing almost all their editing while writing the first draft, and others doing almost all after they have completed the first draft. If modeless editing yields freer editing, we would expect that the *emacs* subjects would tend to do a higher percentage of their editing

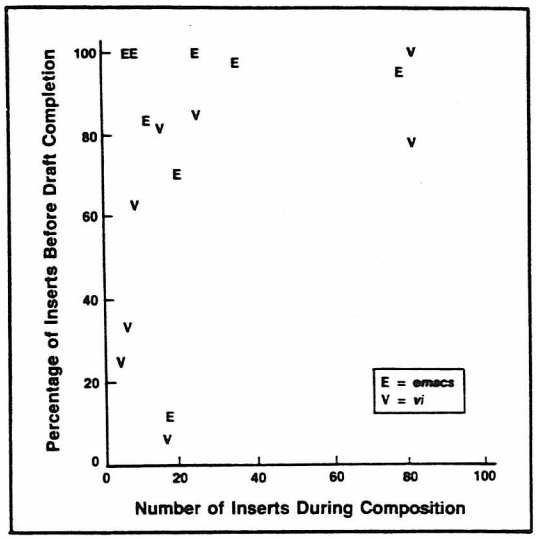

Figure 1. Percentage of Inserts and Changes Before First Draft Completion As a Function of Total Inserts and Changes.

while they were writing their first draft. A Mann-Whitney test on the proportion of edits done during the first draft writing showed that the results supported this prediction ($p = .04$, one-tailed).

Do editors induce differences in editing patterns, or do people choose editors that match their editing patterns? Our feeling is that the differences in editing during composition did not result from people choosing the editor that better matched their patterns of editing. Typically, choice of a screen editor, as revealed both in this study and in our previous interviews, seemed more related to external factors than to editor differences. People reported learning their editors: (1) because they were the only screen editors available, (2) because they were similar to non-available editors the person was familiar with, or (3) because more help (from co-workers or fellow students) was available. No one reported testing and evaluating both editors and then choosing one. Many did report trying one editor after learning the other, but no one switched from one editor to the other as a result of these trials.

There does seem to be some validity to the feeling that modeless editing yields freer authoring. However, the analyses on the writing variables suggest that this freedom may have little effect on the final product or on how long it takes to produce that product.

SUMMARY AND CONCLUSIONS

Experienced *emacs* and *vi* users, who use their editors to write and edit English text, performed a series of basic editing tasks and wrote a movie or book review. Our findings suggest that moded editing, as exemplified in the *vi* editor, may be preferable for fixed editing tasks, while modeless editing, as exemplified by the *emacs* editor, may have some advantages for free composing.

The *vi* subjects left fewer uncorrected errors in their final files than did the *emacs* subjects when doing fixed editing tasks from marked up hard copy. The *emacs* group tended to take longer to complete the editing tasks, but the time differences may have resulted from differences in typing speed.

Moded errors do not seem to be a problem for experienced *vi* users. The *vi* group made few moded errors, and those few were rapidly corrected. Furthermore, modeless editing may not totally avoid moded type errors, since the *emacs* group made errors that were similar in nature to the *vi* moded errors.

Modeless editing may allow people to edit more freely while they are composing. The *emacs* subjects tended to do more of their editing in the first draft stage of composing than did the *vi* group. However, the two groups were similar in total editing, composing time, and most writing style variables.

The ideal editor design would combine moded and modeless editing features. To do this, we suggest adding some basic cursor moving and delete commands using control character names to the inserting text modes. This design provides the editing advantages of moded editing, and, at the same time, allows people who like to edit while composing to do so without having to exit and reenter insertion modes.

This study was not an evaluation of *emacs* or *vi* as total editors. Both are sophisticated editors that provide many features not explored in this study. Although we interpret our results as showing that *vi* was preferable for the fixed editing tasks investigated, we can easily think of cases in which *emacs* would be a far more desirable editor.

REFERENCES

Alle81 Allen, R. B. Composition and editing of text. *Ergonomics,* 1981, *24(8),* 611-622.

Card80 Card, S. K., Moran, T. P., and Newell, A. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM,* 1980 *23(7),* 396-410.

Cher82 Cherry, L. L. Writing tools. *IEEE Transactions on Communications,* 1982, *Com-30(1),* 100-105.

Dzid78 Dzida, H., Herda, S., and Itzfeldt, W. D. User perceived quality of interactive systems. *IEEE Transactions on Software Engineer,* 1978, *SE-4(4),* 270-276.

Good81 Good, M. Etude and the folklore of user interface design. *Sigplan Notices,* 1981, *16(6),* 34-43.

Joy81 Joy, W. *An Introduction to Display Editing with Vi.* Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley, 1981.

Macd82 Macdonald, N. H., Frase, L. T., Gingrich, P. S., and Kennan, S. A. The writer's workbench: computer aids for text analysis. *IEEE Transactions on Communications,* 1982, *Com-30(1),* 105-110.

Meyr82 Meyrowitz, N. and Van Dan, A. Interactive editing systems: part II. *ACM Computing Surveys,* 1982, *14(3),* 353-415.

Newm79 Newman, M. and Sproull, F. *Principles of Interactive Computer Graphics Second Edition,* McGraw-Hill Book Company, New York, 1979.

Robe80 Roberts, T. L. *Evaluation of Computer Text Editors,* Ph.D. dissertation, Department of Computer Science, Stanford University, 1980.

Robe82 Roberts, T. L. and Moran T. P., Evaluation of Text Editors. *Proceedings, 1982 Human Factors in Computer Systems,* 1982, 136-141.

Snee78 Sneeringer, J. User-interface design for text editing; a case study. *Software-Practice and Experience,* 1978, *8,* 543-557.

Stal79 Stallman, R. M. *Emacs: The Extensible, Customizable, Self-Documenting Display Editor.* MIT Artificial Intelligence Laboratory, *AI Memo #519,* Cambridge, Mass., 1979.

Tesl81 Tesler, L. The smalltalk environment. *Byte,* August 1981, 90-147.

Thim82 Thimbleby, H. Character level ambiguity: consequences for user interface design. *Int. J. Man-Machine Studies,* 1982, *16,* 211-225.

Wine71 Winer, B. J. *Statistical Principles in Experimental Design, Second Edition,* McGraw-Hill Book Company, New York, 1971.